Towards More Practical Time-Driven Cache Attacks

Raphael Spreitzer[1,⋆] and Benoît Gérard[2]

[1] IAIK, Graz University of Technology, Austria
raphael.spreitzer@iaik.tugraz.at
[2] DGA-MI, France
benoit.gerard@irisa.fr

Abstract. Side-channel attacks are usually performed by employing the "divide-and-conquer" approach, meaning that leaking information is collected in a *divide step*, and later on exploited in the *conquer step*. The idea is to extract as much information as possible during the *divide step*, and to exploit the gathered information as efficiently as possible within the *conquer step*. Focusing on both of these steps, we discuss potential enhancements of Bernstein's cache-timing attack against the Advanced Encryption Standard (AES). Concerning the *divide part*, we analyze the impact of attacking different key-chunk sizes, aiming at the extraction of more information from the overall encryption time. Furthermore, we analyze the most recent improvement of time-driven cache attacks, presented by Aly and ElGayyar, according to its applicability on ARM Cortex-A platforms. For the *conquer part*, we employ the optimal key-enumeration algorithm as proposed by Veyrat-Charvillon *et al.* to significantly reduce the complexity of the exhaustive key-search phase compared to the currently employed threshold-based approach. This in turn leads to more practical attacks. Additionally, we provide extensive experimental results of the proposed enhancements on two Android-based smartphones, namely a *Google Nexus S* and a *Samsung Galaxy SII*.

Keywords: AES, ARM Cortex-A, key-chunk sizes, optimal key-enumeration algorithm, time-driven cache attack.

1 Introduction

Side-channel attacks have been shown to represent a powerful means of exploiting unintended information leakage on modern system architectures in order to break cryptographic implementations. One specific form of such side-channel attacks is denoted as cache attacks, which aim at the exploitation of different memory-access times within the memory hierarchy. More formally, the central-processing unit (CPU) is able to access data within the cache memory much faster than data within the main memory. These timing differences allow an attacker to break cryptographic implementations [10, 16].

Recent investigations of the timing leakage due to the cache memory, *i.e.*, cache hits and cache misses, emphasized the general applicability of these attacks [2]. However, especially on the ARM Cortex-A platform—the most commonly used architecture in

⋆ This work has been supported by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

modern mobile devices—these investigations [18, 25] showed that timing information is leaking, but the complexity of the remaining key-search phase is usually very high. For instance, Weiß *et al.* [25] compared the vulnerability of different Advanced Encryption Standard (AES) implementations on the ARM platform. For the most vulnerable implementation, *i.e.*, Bernstein's Poly 1305-AES implementation, they presented a remaining key-search complexity of about 65 bits. Such an order of magnitude has been confirmed in [19]. While being clearly within the range of a mafia or state institution—that have far more efficient techniques to recover information anyway—such attacks may be out of reach for hackers and criminals. More precisely, such an attack would require a huge effort that could only be invested for hacking a few users.

Motivation and Contribution. The motivation of this work is to investigate potential improvements of time-driven cache attacks to determine if such attacks could be massively performed by skilled hackers. In this respect, we propose and investigate multiple enhancements to the cache-based timing attack of Bernstein [4] in order to evaluate more accurately the actual security of ARM-based devices regarding this threat.

Bernstein's timing attack is based on the so-called *divide-and-conquer* strategy. While the *divide part* aims at the gathering of the leaking information, *i.e.*, the overall encryption time, the *conquer part* focuses on the actual exploitation of the gathered information to recover the employed secret key. We study potential improvements for both steps. Thus, our contributions can be summarized as follows.

– Regarding the *divide part* we discuss the potential improvement of attacking different key-chunk sizes, *i.e.*, key chunks not corresponding to one byte. Thereby, we try to extract even more information from the observed encryption time under a secret key. Furthermore, we investigate the proposed enhancement of Aly and ElGayyar [2], *i.e.*, the exploitation of the minimum encryption time, according to its applicability on ARM Cortex-A platforms.
– Regarding the context of the *conquer part* we focus on an optimal way to iterate over potential key candidates. While Bernstein initially proposed a *threshold-based approach* to sort out potential key candidates for an exhaustive search, we apply the *optimal key-enumeration algorithm* of Veyrat-Charvillon *et al.* [23]. This allows us to iterate over potential key candidates according to their probability for being the correct one and, thus, to reduce the complexity of the remaining key-search phase.

All discussions about potential improvements are supported by tests performed on two smartphones employing an ARM Cortex-A processor, namely a *Google Nexus S* and a *Samsung Galaxy SII*. These practical experiments demonstrate that the number of key bits to be searched exhaustively can be reduced significantly, hence answering positively the motivating question: timing attacks are within the range of a skilled hacker.

Outline. The remainder of this paper is organized as follows. In Section 2, we cover the required preliminaries including AES software implementations, CPU caches, and cache attacks in general. Section 3 provides the required details of Bernstein's timing attack as well as follow-up work on Bernstein's attack. We consider potential improvements of the *divide part* in Section 4 and discuss the use of the optimal key-enumeration algorithm within the *conquer part* in Section 5. Section 6 states our practical observations of the proposed enhancements. Finally, we conclude this work in Section 7.

2 Background

This section details the basic concept of the Advanced Encryption Standard, the CPU cache memory, as well as the basic principles of cache attacks.

2.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) [12] is a block cipher operating on 128-bit states—denoted as $\mathbf{S} = (\mathbf{s}_0, \ldots, \mathbf{s}_{15})$—and supports key lengths of 128, 192, and 256 bits. The initial state is computed as $\mathbf{S}^0 = \mathbf{P} \oplus \mathbf{K}^0$, with $\mathbf{P} = (\mathbf{p}_0, \ldots, \mathbf{p}_{15})$ being the plaintext and $\mathbf{K}^0 = \left(\mathbf{k}_0^0, \ldots, \mathbf{k}_{15}^0\right)$ the initial round key. After the initial key addition, the round transformations (1) *SubBytes*, (2) *ShiftRows*, (3) *MixColumns*, and (4) *AddRoundKey* are applied multiple times, whereas the last round omits the *MixColumns* transformation. The exact number of rounds depends on the actual key length.

For the purpose of cache attacks the details of these round transformations are mostly irrelevant since software implementations usually employ so-called T-tables, denoted as $\mathbf{T}_i$. These T-tables hold the precomputed round transformations and are composed of 256 4-byte elements. In every round the state bytes $\mathbf{s}_i$ are used to retrieve the precomputed 4-byte values which are then combined with a simple XOR operation to form the new state. The resulting state after the last round represents the ciphertext.

In this work, we consider a key length of 128 bits. However, the outlined concepts shall apply to other key lengths analogously.

2.2 CPU Caches

Since the CPU is not able to access the main memory at the desired speed, the CPU cache has been introduced. The purpose of the CPU cache—a small and fast memory between the CPU and the main memory—is to hold close frequently used data and, thus, to enhance the performance of memory accesses. The most commonly used caches are so-called *set-associative caches* where the cache is divided into equally sized cache sets, each consisting of multiple cache lines. Contiguous bytes of the main memory are then mapped to a specific cache set and can be placed in any cache line of this cache set. The actual cache line within a cache set where new data should be placed is determined by the replacement policy, which can be either random or deterministic. In case of the ARM Cortex-A platform a random-replacement policy is employed.

2.3 Cache Attacks

Combining the knowledge about T-table implementations and CPU caches leads to the concept of cache attacks. First, AES T-table implementations make use of key-dependent look-up indices to access the precomputed values of the round transformations. Second, these T-table accesses are not performed in constant time. The data might be fetched either from the CPU cache (cache hit) or from the main memory (cache miss) and, thus, leads to varying execution times. Hence, cache attacks are a specific form of side-channel attacks that aim at the exploitation of variations within the execution time.

Cache attacks are separated into three categories: (1) *time-driven attacks*, (2) *access-driven attacks*, and (3) *trace-driven attacks*. Time-driven cache attacks [4, 22] rely on the overall encryption time to recover the used secret key by means of statistical methods. Hence, these attacks represent the most general type of cache attacks. In contrast, access-driven cache attacks [9, 14, 20, 21] as well as trace-driven cache attacks [1, 6, 7] rely on more sophisticated knowledge about the implementation and the underlying hardware architecture. However, access-driven and trace-driven attacks require far less measurement samples than time-driven attacks. In short, there is a trade-off between the required knowledge and the number of required measurement samples. In this work, we focus on the investigation of the time-driven cache attack proposed by Bernstein [4].

3 Related Work

In this section, we detail the basic concept of Bernstein's timing attack [4] and outline related work based on this attack.

3.1 Seminal Work: Bernstein's Timing Attack

In 2005, Bernstein [4] proposed a timing attack against the AES T-table implementation. He suggested to gather timing information of different plaintexts under a known key $\mathbf{K}$ as well as under an unknown key $\widetilde{\mathbf{K}}$. Afterwards, correlating the timing information of these two sets of plaintexts should reveal potential key candidates. The attack consists of four different phases which are outlined within the following paragraphs.

Study Phase. Within this phase the attacker measures the encryption time of multiple plaintexts $\mathbf{P}$ under a known key $\mathbf{K}$. Without loss of generality, we assume that the zero-key is used for this phase. The information is stored in $\mathbf{t}[j][b]$ which holds the sum of all encryption times observed for plaintexts where the plaintext byte $\mathbf{p}_j = b$, and $\mathbf{n}[j][b]$ which counts the number of encrypted plaintexts where $\mathbf{p}_j = b$.

Attack Phase. In this phase the attacker collects the exact same information as in the study phase, but this time under an unknown key $\widetilde{\mathbf{K}}$ that she wants to recover. The gathered information is stored in $\widetilde{\mathbf{t}}[j][b]$ and $\widetilde{\mathbf{n}}[j][b]$, respectively.

Correlation Phase. The attacker computes the so-called plaintext-byte signature [13] of the study phase as illustrated in Equation 1. The plaintext-byte signature of the attack phase is computed analogously, except that $\mathbf{v}[j][b]$, $\mathbf{t}[j][b]$, and $\mathbf{n}[j][b]$ are replaced with $\widetilde{\mathbf{v}}[j][b]$, $\widetilde{\mathbf{t}}[j][b]$, and $\widetilde{\mathbf{n}}[j][b]$, respectively.

$$\mathbf{v}[j][b] = \frac{\mathbf{t}[j][b]}{\mathbf{n}[j][b]} - \frac{\sum_j \sum_b \mathbf{t}[j][b]}{\sum_j \sum_b \mathbf{n}[j][b]} \tag{1}$$

Afterwards, the correlations of the plaintext-byte signature within the study phase and the attack phase are computed as outlined in Equation 2.

$$\mathbf{c}[j][b] = \sum_{i=0}^{255} \mathbf{v}[j][i] \cdot \widetilde{\mathbf{v}}[j][i \oplus b] \tag{2}$$

The correlations are sorted in a decreasing order and, based on a predefined threshold, the attacker obtains a list of potential values for each key byte $\mathbf{k}_j$.

Key-Search Phase. Usually, more than one value per byte is selected in the correlation phase. Thus, the attacker performs an exhaustive search over all possible key candidates that can be formed from the selected values using a known plaintext-ciphertext pair.

3.2 Applications and Improvements of Bernstein's Attack

Bernstein's idea of exploiting the timing leakage of T-table based AES implementations has gained particular attention among the scientific community. For instance, Neve [13] and Neve *et al.* [15] performed a detailed analysis of Bernstein's timing attack. In 2012, Weiß *et al.* [25] compared the vulnerability of different AES implementations. Spreitzer and Plos [18] investigated the applicability of time-driven cache attacks, including the one of Bernstein, on mobile devices. Aly and ElGayyar [2] also investigated this timing attack and introduced an additional timing information, which is used to correlate the timing profiles obtained in the study phase with the timing profiles obtained in the attack phase. This timing information consists of the overall minimum encryption time (*global minimum*) and the minimum encryption time for a specific plaintext byte at a specific position (*local minimum*). Recently, Saraswat *et al.* [17] investigated the applicability of Bernstein's timing attack against remote servers.

4 Analysis and Improvements of the Divide Part

In this section, we detail potential improvements regarding the gathering of the required timing information. We present the corresponding experimental results in Section 6.

4.1 Attacking Different Key-Chunk Sizes

While Bernstein [4] considered the leaking timing information for exactly one key byte, we investigate a potential improvement of this attack by considering the leaking timing information of different key-chunk sizes. To this end, we briefly analyze the main concept of this idea as well as potential pitfalls.

Let n_{kc} be the number of key chunks the whole key is comprised of, and s_{kc} be the size of each key chunk, *i.e.*, the number of possible values each key chunk might take. If we attack each key byte separately ($n_{kc} = 16$, $s_{kc} = 256$), then $\mathbf{t}[j][b]$ holds the total of all encryption times where plaintext byte $\mathbf{p}_j = b$ and $\mathbf{n}[j][b]$ counts the number of plaintexts where $\mathbf{p}_j = b$, for $0 \leq j < n_{kc}$ and $0 \leq b < s_{kc}$. Hence, attacking larger parts of the key at once leads to fewer key chunks n_{kc}, but a larger number of possible values per key chunk s_{kc}. In contrast, attacking smaller parts of the key leads to a larger number of key chunks n_{kc} with fewer possible values s_{kc} for each of these key chunks.

Considering the plaintext as a 4×4 matrix, we observe that larger blocks can be formed in different ways. For instance, Figure 1 illustrates the gathering of the timing information for two consecutive plaintext bytes of one specific row. In contrast, Figure 2 illustrates the combination of two plaintext bytes within one specific column. In case of the T-table implementation provided by OpenSSL the former approach collects timing information of two bytes accessing two different T-tables and the latter collects timing information of two bytes accessing the same T-table. For the practical evaluation we

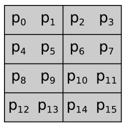

Fig. 1. Combination of bytes within a row

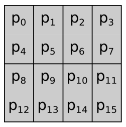

Fig. 2. Combination of bytes within a column

implemented the approach that collects timing information of two bytes accessing the same T-table. This case reduces the eventuality of corrupting key chunks with noise that might affect a specific T-table. We illustrate this within the following example. Recall that look-up indices $\mathbf{s}_i$ access T-table $\mathbf{T}_j$, with $i \equiv j \mod 4$. Now, suppose that noise affects T-table $\mathbf{T}_0$. Then in case of attacking two-byte key chunks that access the same T-table (*e.g.*, $\mathbf{T}_0$), only two key chunks are affected by this noise. In contrast, if we attack two-byte key chunks that access different T-tables (*e.g.*, $\mathbf{T}_0$ and $\mathbf{T}_1$), four key chunks are affected by this noise.

Within the following paragraphs we investigate potential pitfalls of attacking different key-chunk sizes. Corresponding experimental results can be found in Section 6.

Memory Requirements. The memory consumption of the timing attack depends on the number of key chunks n_{kc}, the size of each key chunk s_{kc}, and the size s_d of the employed data type in bytes. Assuming the size of the data type $s_d = 8$, then for attacking each key byte separately $(n_{kc} = 16, s_{kc} = 256)$ the size of one such data structure is 32 KB. Attacking two bytes at once $(n_{kc} = 8, s_{kc} = 256^2)$ would result in 4 MB for each data structure and attacking even four bytes at once $(n_{kc} = 4, s_{kc} = 256^4)$ would result in 128 GB for each data structure. Thus, attacking more than two bytes at once is not applicable for devices with limited resources, *e.g.*, mobile devices.

Number of Measurement Samples. The key-chunk size also has an impact on the noise reduction of the timing information. First, notice that the larger the key chunks are, the smaller should be the algorithmic noise in the gathered encryption times. This positive effect of large chunks is counterbalanced by the fact that the larger the chunks are, the smaller is the number of samples obtained for a specific chunk value[1]. Thus, there is a trade-off between the algorithmic noise and the number of samples per value. Analyzing this requires the full understanding of the noise behavior (what is the proportion of algorithmic noise compared to measurement noise and what is the noise distribution) and is out of the scope of this paper. We thus ran experiments to get an empirical evidence for the best trade-off.

Indistinguishable Key Bits. The cache-line size determines the number of contiguous T-table elements to be loaded at once in the event of a cache miss. In case of a cache-line size of 32 (resp. 64) bytes, each cache line holds 8 (resp. 16) T-table elements. This in turn means that in general one cannot distinguish between accessed elements in one specific cache line and, therefore, the number of recoverable key bits is limited. Let

[1] Let N be the number of encrypted plaintexts, then each possible value b of a specific block $\mathbf{p}_j$ is encrypted approximately $\frac{N}{s_{kc}}$ times.

us denote by s_c the cache-line size in bytes and by s_t the size of a T-table element in bytes. Then, according to Tromer *et al.* [21] the number of non-recoverable key bits per key byte—at least for attacks considering only the first round of the AES—is given as $\log_2 \frac{s_c}{s_t}$. This means that in case of a cache-line size of 32 (resp. 64) bytes the number of non-recoverable key bits per key byte are 3 (resp. 4). However, this is just a theoretical observation since in practice more advanced features of the architecture like *critical word first*[2] or *early restart*[3], as well as particular properties of the implementation itself, *i.e.*, disaligned T-tables, usually lead to more information leakage.

4.2 Template-Attack Approach

Template attacks [5] are optimal attacks if the attacker gets perfect knowledge of the leakage distribution on the targeted device. The idea is to make a template of the leakage, *i.e.*, computing the leakage distribution in a known-key setting, and then performing an attack by exploiting leakages and the knowledge of the computed distribution.

The templates can be made (i) by observing a twin device owned (and controlled) by the attacker or (ii) by using the target device if the attacker is able to observe encryptions performed with a known key. In both cases the attacker expects that the characterization she made during the learning phase will still be valid during the attack.

4.3 Minimum Timing Information

Recently, Aly and ElGayyar [2] suggested to compute the correlation of the minimum encryption time within the study phase and the minimum encryption time within the attack phase. Therefore, $\mathbf{tmin}$ and $\widetilde{\mathbf{tmin}}$ hold the overall minimum encryption time of all encrypted plaintexts in the study phase and the attack phase, respectively. In addition, the data structure $\mathbf{umin}[j][b]$ holds the minimum encryption time of all plaintexts $\mathbf{p}$ where $\mathbf{p}_j = b$. The same holds for the attack phase, except that $\mathbf{umin}[j][b]$ and $\mathbf{p}$ are replaced by $\widetilde{\mathbf{umin}}[j][b]$ and $\widetilde{\mathbf{p}}$, respectively. The computation of the correlation is based on $\mathbf{umin}[j][b] - \mathbf{tmin}$ and $\widetilde{\mathbf{umin}}[j][b] - \widetilde{\mathbf{tmin}}$. Combining the timing information initially proposed by Bernstein and the minimum timing information, they claim to recover the whole secret key without a single exhaustive key-search computation. We assume that the attacker computes the correlation with the timing information initially proposed by Bernstein and the correlation with the minimum timing information. So for each key byte $\mathbf{k}_i$ the attacker retrieves two sets of potential key candidates. Afterwards the attacker combines the sets of potential key candidates with the lowest cardinalities.

5 Analysis and Improvements of the Conquer Part

The main challenge of the conquer part is to gather information obtained during the divide step to recover the full key.

[2] *Critical word first* means that in case of a cache miss the missed word is loaded first and then the CPU continues its work while the remaining words are loaded into the cache.

[3] *Early restart* means that as soon as the critical word arrives, the CPU continues its work. In practice this would impose a serious side channel.

5.1 Combining Information from the Divide Part

We briefly describe two possible approaches to recover the full key. The first one being the one currently used in timing attacks, and the second one overcoming some of the mentioned shortcomings of the first one.

Threshold Approach. Currently, timing attacks employ a threshold-based approach. This means that one fixes a threshold on the computed correlations and considers sub-key values as potential candidates only if the corresponding correlation is larger than the threshold. Notice that one may use different thresholds for the different sub-keys, either because a profiling phase has shown different behaviors for different sub-keys or because they are dynamically computed.

The threshold approach is simple to implement but has two major drawbacks. The first one is that the actual key may not be found. Indeed, if one of its sub-key values led to a small correlation, then the key will never be tested in the search-phase and, thus, the attack will provide no advantage over exhaustive search. The second drawback is a loss of information since the ordering of kept sub-key values is not exploited in the search phase. Though Neve [13, p 58] suggested that the key search "could start by the most probable key candidates", no clear indication is given how this should be accomplished. A somehow related approach has been suggested by Meier and Staffelbach [11] in 1991. However, they do not iterate over potential keys which are sorted according to their probability for being the correct one, but instead they exploit the non-uniform probability distribution of the key source. Thus, they generate the keys to be tested from the actual probability distribution.

Optimal Enumeration Approach. Veyrat-Charvillon *et al.* [23] recently proposed an optimal key-enumeration algorithm that solves the aforementioned problems at the cost of additional computations for generating the next full-key to be tested. The algorithm requires a *combination function* that computes the score of the concatenation of two key chunks based on the scores of each chunk. Using such a combination function, a global score can be computed for each full key by combining the sub-key scores. The "optimal" notion comes from the fact that the algorithm ensures that keys will be generated in a decreasing order of global scores.

5.2 Evaluating the Key-Search Complexity

Threshold Approach. The lower bound on the key-search complexity is easy to obtain. Assuming that the attacker dynamically chooses a threshold for each targeted sub-key, it will, for a given sub-key, keep at least all values with a score larger than the correct one. The cardinality of the set of keys to be tested is then equal to the product of sub-key ranks. This lower bound is very optimistic as no such magic threshold choice exists.

Concerning the upper bound, it will depend on the allowed probability of missing the correct key. For given threshold(s), upper bounds on the key-search complexity and estimates of the missing-key probability can be obtained by simulating attacks. The upper bound being the size of the key-search space and the success probability being the probability that the actual key belongs to this space.

Optimal Enumeration Approach. Following the proposition of a key-enumeration algorithm in [23], Veyrat-Charvillon *et al.* [24] proposed a key-rank estimation algorithm that bound the key-search complexity of the optimal key-enumeration algorithm for a given combination function. More precisely, their algorithm requires the combination function, the scores obtained for one attack and the correct key. When stopped, the program provides an interval $[2^x; 2^{x+\epsilon}]$ ensuring that the key rank lies in this range.

5.3 Choosing Relevant Thresholds and Combination Functions

Choosing Thresholds. Thresholds are potentially based on any kind of statistical value and provide a simple means of sorting out potential key candidates. Therefore, after performing the correlation of the timing information gathered within the study phase and the timing information gathered within the attack phase one retrieves a correlation vector $\mathbf{c}[j][b]$ (see Section 3.1). These elements are sorted in a decreasing order and byte values b with a correlation value above a predefined threshold are considered to represent potential key candidates. For instance, Bernstein suggested a threshold which is based on the *standard error of the mean*. By iterating over all possible key values b, the idea is to take a key candidate b for a key byte $\mathbf{k}_j$ only into consideration if the difference of the byte value providing the highest correlation and the correlation of b is smaller than the established threshold.

Choosing Combination Functions. From an information theoretical point of view the optimal choice for a combination function is to turn scores into sub-key probabilities and then combining these probabilities by multiplication. The so-called "Bayesian extension" in [23] uses Bayes' relation and a theoretical model of obtained scores to compute sub-key probabilities. This technique requires the attacker to model scores obtained to be able to estimate relevant probabilities. A recent work on side-channel collision attacks by Gérard and Standaert [8] has shown that even if the model does not accurately match reality, the use of a Bayesian extension may improve attacks.

In the context of timing attacks the scores obtained are similar to correlations. Actually a small modification of the scoring function as defined by Bernstein turns the scores into actual correlations without modifying the ordering of sub-key values. The modified formula for computing scores according to Pearson's correlation coefficient is outlined in Equation 3. The equation is given for an arbitrary key-chunk size s_{kc}.

$$\mathbf{c'}[j][b] = \frac{\sum_{i=0}^{s_{kc}-1} \mathbf{v}[j][i] \cdot \widetilde{\mathbf{v}}[j][i \oplus b]}{\sqrt{\sum_{i=0}^{s_{kc}-1} \mathbf{v}[j][i]^2} \cdot \sqrt{\sum_{i=0}^{s_{kc}-1} \widetilde{\mathbf{v}}[j][i \oplus b]^2}} \tag{3}$$

Then, using a Bayesian extension similar to the one in [8] (based on Fisher transform of correlation coefficients) we are able to estimate sub-key probabilities. The idea is that $\operatorname{arctanh}(\mathbf{c'}[j][b])$ follows a Gaussian distribution of variance $\frac{1}{s_{kc}-3}$ and with mean 1 if $\mathbf{k}_j = b$ and 0 otherwise. The estimated likelihood ratio between the probabilities of the j-th key chunk to be equal to b or not is then:

$$\begin{aligned}\mathbf{l}[j][b] &= \exp\left[\frac{(s_{kc}-3)(\operatorname{arctanh}(\mathbf{c}[j][b]) - 0)^2}{2} - \frac{(s_{kc}-3)(\operatorname{arctanh}(\mathbf{c'}[j][b]) - 1)^2}{2}\right], \\ &= \exp\left[(s_{kc}-3)\left(\operatorname{arctanh}(\mathbf{c'}[j][b]) - 0.5\right)\right].\end{aligned}$$

Table 1. Device specifications for the test devices

Device	Processor	L1 Cache				Critical Word First	OS
		Size	Associativity	Line Size	Sets		
Google Nexus S	Cortex-A8	32 KB	4 way	64 bytes	128	yes	Android 2.3.6
Samsung Galaxy SII	Cortex-A9	32 KB	4 way	32 bytes	256	yes	Android 2.3.4

The score of a full-key candidate **k** will be given by

$$S_{\mathrm{Bayes}} = \prod_j \mathbf{l}[j][\mathbf{k}_j]. \tag{4}$$

To investigate the relevance of such Bayesian extension, Section 6 also contains data obtained with a different combination function that does not use Fisher transform. Natural combinations of correlation coefficients are operators $+$ and $\times$. The latter one is not relevant here since two values with correlation -1 will combine to a key with correlation 1 what is not desirable. We thus propose results obtained using $+$ as a combination function to complement our study. In that case, the score of a full-key candidate **k** will be given by

$$S_{Add} = \sum_j \mathbf{c'}[j][\mathbf{k}_j]. \tag{5}$$

Remark on the Threshold Attack Lower Bound. In [24] authors perform experiments using the output of simulated template attacks. In this context the probabilities that are computed are sound (*i.e.*, not biased by a potentially wrong model) and thus the attack using key enumeration is optimal from an information theoretic point of view. A figure shows that the lower bound obtained by multiplying sub-key ranks (*i.e.*, the lower bound for threshold attacks) is far too optimistic by orders of magnitude from the actual key rank. The optimality of the attack regarding information theory implies that an attacker using threshold technique should not obtain better results and hence this experiment discards this lower bound as a relevant statistic.

6 Experimental Results

In this section, we detail the employed measurement setup and later on we analyze our observations regarding the practical evaluation. For the practical investigation of the suggested enhancements we employed two Android-based smartphones, namely a *Google Nexus S* and a *Samsung Galaxy SII*. The specifications of these two devices are summarized in Table 1. One assumption for our attack to work is that the device under attack must be rooted in order to allow for precise timing measurements via the ARM *Cycle Counter Register* [3].

Definitions. We define the gathering of the measurement samples under a known key and the gathering of the measurement samples under an unknown key as one *run*. Thus, one *run* of the attack application constitutes the gathering of the measurement samples for both of these phases. The *number of measurement samples* denotes the number of gathered samples in each of these two phases.

6.1 Attacking Different Key-Chunk Sizes

We launched the attack multiple times on both devices, targeting either four bits, one byte, or two bytes of the key. Figure 3 shows the rank evolution for a specific number of measurement samples on the *Samsung Galaxy SII*, averaged over multiple runs for one-byte chunks and two-byte chunks, respectively. More formally, these plots show the range (bounds) of key bits to be searched with the optimal key-enumeration algorithm after gathering a specific number of measurement samples. Our observations show that below 2^{21} measurement samples hardly any information leaks. Targeting four-bit chunks we observed a similar rank evolution as for one-byte chunks. Hence, we omitted this figure here.

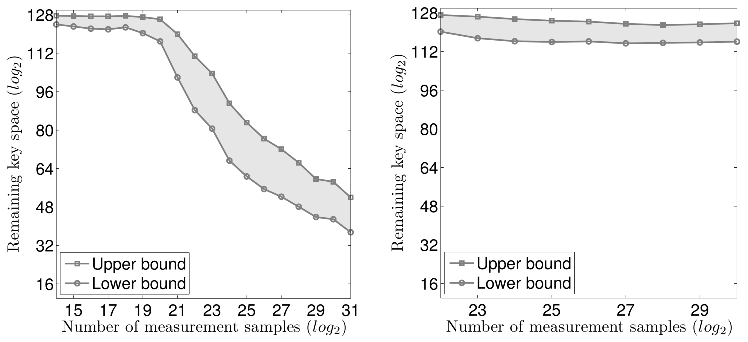

Fig. 3. Rank evolution for one-byte key chunks (left) and two-byte key chunks (right)

According to the right part of Figure 3, the noise induced by the small number of samples per chunk value is significantly larger than the noise reduction obtained by considering larger chunks, which might be due to the random-replacement policy. The problem can be illustrated as follows. Figure 4 shows the plaintext-byte signature for one specific key byte during the study phase and the attack phase, respectively. The abscissa shows the possible chunk values of a plaintext byte and the ordinate shows the average encryption time for this specific byte subtracted by the overall average encryption time, after gathering 2^{30} samples. We observe a visible pattern in both plots. Thus, the correlation yields a few possible values for this specific key byte. We also point out that most of the values lie in the range $[-0.5; 0.5]$ with peaks up to 2.5.

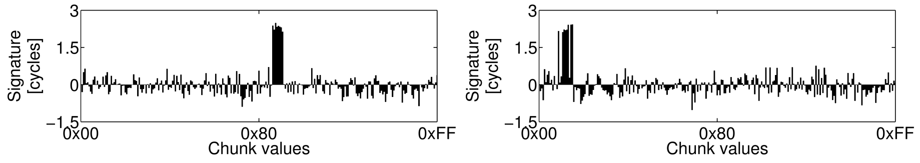

Fig. 4. One-byte chunk signatures for the study phase (left) and the attack phase (right)

In contrast, Figure 5 illustrates the chunk signatures for an attack targeting two-byte key chunks. Again, after gathering 2^{30} measurement samples. Since the pattern is not that clearly visible we marked the similar peaks appropriately. Neve [13] also performed an investigation of such signature plots for one-byte chunks. In accordance with his terminology, we note that both plots show rather *noisy* profiles with most values lying in the range $[-25; 25]$. Due to these *noisy* profiles the correlation does not reduce the key space significantly and the sub-key value for this specific key chunk cannot be determined. Though we also observed rather *noisy* profiles for attacks targeting one-byte chunks, most of the profiles established for one-byte chunks showed a clear pattern. In contrast, for two-byte key chunks we mostly observed plots where we could not find any specific pattern.

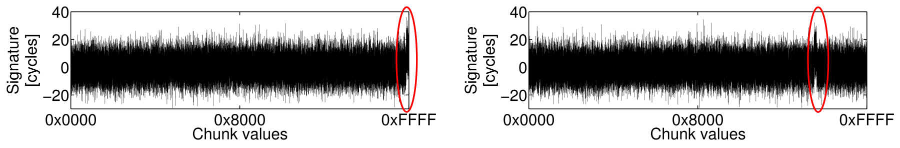

Fig. 5. Two-byte chunk signatures for the study phase (left) and the attack phase (right)

To conclude, our observations showed that attacking smaller key chunks potentially works, while attacking larger key chunks seems to leak less information for the same (realistic) number of measurement samples. Targeting even more samples is not realistic anymore, at least for mobile devices. This results from the fact that a running time of more than eight hours to gather more than 2^{30} measurement samples does not allow for a realistic scenario anymore.

6.2 Template Attack

As a first step we tried to identify the distribution of encryption times. Obviously the classical Gaussian noise model (that is quite relevant for power-based attacks) does not fit here. Moreover, the right tail of the distribution is meaningless since high encryption times are caused by interruptions. The choice of a threshold above which we consider points as outliers together with the characterization of the distribution of remaining points is not straightforward and out of the scope of the paper.

Noting the difficulty of characterizing the time distribution, we mount attacks combining the study phase and the attack phase from two different runs (that is the data sets have not been measured one after the other). We used the key-rank estimation (and the Bayesian extension as detailed in Section 5.3) to estimate the remaining workload for the key-search phase. We observe that we obtain ranks 2^{10} larger than the one obtained from attacks where the attack phase directly follows the study phase.

One reason for this observation might be the fact that ARM Cortex-A series processors employ a physically-indexed, physically-tagged (PIPT) data cache, which means that the physical address is used to map a location within the main memory to a cache set. For different runs the physical address potentially changes and, thus, the locations where memory accesses (resulting in cache evictions) occur change from run to run.

6.3 Minimum Timing Information

Aly and ElGayyar [2] argue that noise usually increases the encryption time and, thus, the exploitation of the minimum timing information should significantly improve the timing attack. Their idea is to capture only one single measurement sample without noise, which is then stored and used for the correlation later on. They successfully launched their attack against a *Pentium Dual-Core* and a *Pentium Core 2 Duo* processor. However, contrary to their conclusion that this approach significantly improves the timing attack on Pentium processors, our results indicate that this approach does not even work at all on ARM Cortex-A processors. The reasons for this approach to fail on the ARM Cortex-A processor are potentially manifold. First, Aly and ElGayyar [2] attacked an AES implementation employing 4-Tables. In contrast, we attacked an implementation employing 5 T-tables.

Second, according to our understanding, gathering the minimum encryption time misses potential useful information. As Neve *et al.* [15] put it, Bernstein's timing attack implicitly searches for cache evictions due to work done on the attacked device. Such cache evictions lead to cache misses within the encryption function and, thus, to slower encryptions. As a result, not only noise increases the encryption time, but also cache misses increase the encryption time. While noisy encryption times do not carry useful information, encryption times where a cache miss occurred definitely do so. However, gathering the minimum timing information does not capture this information because the minimum timing information seeks for encryption times where a cache hit occurred. The problem is that once we observe a cache hit, *i.e.*, a fast encryption, we store this timing information. So this approach only searches for the cache hits and in the worst case, after a certain number of measurement samples, we potentially observe a cache hit for all possible key bytes due to the random-replacement policy on ARM processors. Furthermore, in the long run, the *local minimum* as well as the *global minimum* might become equal in which case these timings do not carry any information at all. Our practical evaluation showed that after a certain number of measurement samples on the *Google Nexus S* the minimum timing information $\mathbf{umin}[j][b]-\mathbf{tmin}$ equals 0 for most of the key bytes. Additionally, the random-replacement policy employed in ARM Cortex-A processors strengthens this reasoning. Though Aly and ElGayyar implemented this approach on Pentium processors with a deterministic replacement policy, we consider the gathering of the minimum timing information also risky on such processors.

Concluding the investigation of the minimum timing information we point out that instead of using the minimum timing information, we stick to the exploitation of the timing information as proposed by Bernstein and only take encryption times below a specific threshold into consideration. This approach also reduces the impact of noise if the threshold is selected properly.

Table 2. Sample results on the Samsung Galaxy SII

Run	Key-Chunk Size	Samples	Bernstein				Minimum
			Optimal Threshold	Threshold	Key Enumeration		Optimal Threshold
					(4)	(5)	
1	4 bits	2^{30}	50 bits	102 bits	79.3 - 104.4	86.5 - 112.3	84 bits
2	4 bits	2^{31}	32 bits	87 bits	58.9 - 82.4	62.1 - 92.6	88 bits
3	1 byte	2^{28}	41 bits	93 bits	55.7 - 77.7	56.2 - 79.2	104 bits
4	1 byte	2^{30}	23 bits	64 bits	36.6 - 44.9	36.4 - 46.5	100 bits
5	1 byte	2^{30}	32 bits	92 bits	49.1 - 70.1	49.1 - 70.3	100 bits
6	1 byte	2^{30}	20 bits	74 bits	36.5 - 45.6	36.0 - 46.4	105 bits
7	2 bytes	2^{30}	107 bits	123 bits	118.9 - 125.3	118.9 - 125.3	104 bits
8	2 bytes	2^{30}	96 bits	128 bits	115.5 - 122.2	115.0 - 123.2	114 bits
9	2 bytes	2^{30}	90 bits	124 bits	110.5 - 119.7	110.5 - 119.9	118 bits
10	2 bytes	2^{30}	110 bits	126 bits	120.2 - 126.7	120.3 - 126.7	115 bits

6.4 Summary of Practical Results

Table 2 summarizes the results of our practical investigations on the *Samsung Galaxy SII* smartphone. For different runs, we provide the attacked key-chunk size as well as the number of samples acquired. The rest of the columns contain different $\log_2$ time complexities of the key-search phase depending on the exploited information, *e.g.*, either Bernstein's timing information or the minimum timing information from [2], and depending on the conquer-phase technique. For the threshold-based conquer phase we provide the remaining key space for: (1) an optimal threshold choice, such that for each chunk the threshold is chosen in a way that only values with better scores than the correct one are selected (cf. Section 5.2), and (2) a threshold based on the *standard error of the mean* as suggested by Bernstein [4]. The key-enumeration column contains bounds of the obtained key rank if the optimal key-enumeration algorithm from [23] is used. In Table 2 this column is separated into two, the first one being the result of the use of the Bayesian extension (see Equation 4) the second being obtained by addition of correlations (see Equation 5). We clearly observe that using the optimal key-enumeration algorithm instead of the threshold-based approach has a strong positive impact on the key-search complexity. For instance, in case of run 4 and run 6—that require far more than 2^{60} keys to be tested in case of the threshold-based approach—the optimal key-enumeration algorithm recovers the key in less than 2^{46} tests. Concerning the gained improvement of using the Bayesian extension, we observe that it is very small when attacking one-byte chunks but becomes more significant when attacking four-bit chunks.

Furthermore, for ARM Cortex-A processors we cannot confirm that the minimum timing information improves the timing attack. The last column in Table 2 shows that this information hardly leaks any information. Table 3 summarizes the exact same information for the *Google Nexus S* smartphone. Since we observed only minor differences between the usage of the Bayesian extension (Equation 4) and the usage of addition as a correlation function, we only provide the bounds based on the former approach.

7 Conclusion

In this work, we analyzed multiple improvements of Bernstein's timing attack. Considering these improvements we also provided practical insights on two devices employing an ARM Cortex-A processor. We performed theoretical investigations of attacking

Table 3. Sample results on the Google Nexus S

Run	Key-Chunk Size	Samples	Bernstein			Minimum
			Optimal Threshold	Threshold	Key Enumeration	Optimal Threshold
1	1 byte	2^{31}	64 bits	108 bits	83.5 - 109.1 bits	105 bits
2	1 byte	2^{30}	62 bits	119 bits	77.6 - 104.9 bits	95 bits
3	1 byte	2^{26}	66 bits	101 bits	79.0 - 101.3 bits	104 bits
4	1 byte	2^{30}	67 bits	96 bits	77.6 - 104.4 bits	108 bits
5	1 byte	2^{30}	58 bits	91 bits	69.6 - 90.5 bits	107 bits
6	1 byte	2^{28}	82 bits	95 bits	105.3 - 115.2 bits	110 bits
7	1 byte	2^{30}	61 bits	97 bits	84.0 - 99.0 bits	97 bits
8	2 bytes	2^{27}	121 bits	128 bits	127.8 - 128.0 bits	121 bits
9	2 bytes	2^{28}	116 bits	128 bits	125.0 - 127.8 bits	124 bits
10	2 bytes	2^{30}	118 bits	128 bits	126.1 - 127.2 bits	121 bits

different key-chunk sizes and presented potential pitfalls. Our practical investigations on ARM-based devices showed that attacking one-byte chunks seems to be the best choice for resource-constrained devices. Furthermore, these investigations showed that the minimum timing information [2] does not improve the cache timing attack on the ARM Cortex-A devices at all. We also showed that due to the PIPT data cache of ARM Cortex-A processors, template attacks seem to be useless here. Nevertheless, a thorough analysis of the noise behavior might lead to more positive results. We let this point as an open question.

The most important contribution of this work is the shift from the threshold-based approach for the selection of potential key candidates towards the application of the optimal key-enumeration algorithm. Instead of selecting potential key candidates on a threshold basis, we iterate over potential keys according to their probability for being the correct key. As our observations showed, this approach significantly reduces the complexity of the remaining key-search phase, which brings this attack to a complexity that can be considered as practically relevant.

References

[1] O. Aciiçmez and Çetin Kaya Koç. Trace-Driven Cache Attacks on AES (Short Paper). In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *LNCS*, pages 112–121. Springer, 2006.

[2] H. Aly and M. ElGayyar. Attacking AES Using Bernstein's Attack on Modern Processors. In A. Youssef, A. Nitaj, and A. E. Hassanien, editors, *AFRICACRYPT*, volume 7918 of *LNCS*, pages 127–139. Springer, 2013.

[3] ARM Ltd. *ARM Technical Reference Manual, Cortex-A8, Revision: r3p2*, May 2010.

[4] D. J. Bernstein. Cache-timing attacks on AES. Available online at `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[5] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[6] J.-F. Gallais and I. Kizhvatov. Error-Tolerance in Trace-Driven Cache Collision Attacks. In *COSADE*, pages 222–232, Darmstadt, 2011.

[7] J.-F. Gallais, I. Kizhvatov, and M. Tunstall. Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. In Y. Chung and M. Yung, editors, *WISA*, volume 6513 of *LNCS*, pages 243–257. Springer, 2010.

[8] B. Gérard and F.-X. Standaert. Unified and Optimized Linear Collision Attacks and their Application in a Non-Profiled Setting: Extended Version. *J. Cryptographic Engineering*, 3(1):45–58, 2013.

[9] D. Gullasch, E. Bangerter, and S. Krenn. Cache Games - Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy*, pages 490–505. IEEE Computer Society, 2011.

[10] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *ESORICS*, volume 1485 of *LNCS*, pages 97–110. Springer, 1998.

[11] W. Meier and O. Staffelbach. Analysis of Pseudo Random Sequence Generated by Cellular Automata. In D. W. Davies, editor, *EUROCRYPT*, volume 547 of *LNCS*, pages 186–199. Springer, 1991.

[12] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001.

[13] M. Neve. *Cache-based Vulnerabilities and SPAM Analysis*. PhD thesis, UCL, 2006.

[14] M. Neve and J.-P. Seifert. Advances on Access-Driven Cache Attacks on AES. In E. Biham and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *LNCS*, pages 147–162. Springer, 2006.

[15] M. Neve, J.-P. Seifert, and Z. Wang. A refined look at Bernstein's AES side-channel analysis. In F.-C. Lin, D.-T. Lee, B.-S. P. Lin, S. Shieh, and S. Jajodia, editors, *ASIACCS*, page 369. ACM, 2006.

[16] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. *IACR Cryptology ePrint Archive*, 2002:169, 2002.

[17] V. Saraswat, D. Feldman, D. F. Kune, and S. Das. Remote Cache-timing Attacks Against AES. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, CS2 '14, pages 45–48, New York, NY, USA, 2014. ACM.

[18] R. Spreitzer and T. Plos. On the Applicability of Time-Driven Cache Attacks on Mobile Devices. In J. Lopez, X. Huang, and R. Sandhu, editors, *Network and System Security*, volume 7873 of *LNCS*, pages 656–662. Springer Berlin Heidelberg, 2013.

[19] R. Spreitzer and T. Plos. On the Applicability of Time-Driven Cache Attacks on Mobile Devices (Extended Version). *IACR Cryptology ePrint Archive*, 2013:172, 2013.

[20] J. Takahashi, T. Fukunaga, K. Aoki, and H. Fuji. Highly Accurate Key Extraction Method for Access-Driven Cache Attacks Using Correlation Coefficient. In C. Boyd and L. Simpson, editors, *ACISP*, volume 7959 of *LNCS*, pages 286–301. Springer, 2013.

[21] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptology*, 23(1):37–71, 2010.

[22] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In C. D. Walter, Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2779 of *LNCS*, pages 62–76. Springer, 2003.

[23] N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F.-X. Standaert. An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *LNCS*, pages 390–406. Springer, 2012.

[24] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert. Security Evaluations beyond Computing Power. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 126–141. Springer, 2013.

[25] M. Weiß, B. Heinz, and F. Stumpf. A Cache Timing Attack on AES in Virtualization Environments. In A. D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *LNCS*, pages 314–328. Springer, 2012.